Mobile and Pervasive Computing 23/24

Project Checkpoint – BrightBalance+

Authors: 64783 André Correia
64813 Rodrigo Fontinha
68759 Tiago Martinho

May 13th, 2024

Description

The idea behind this project originated from the human necessity to have a conscious and respectful attitude towards the use of everyday resources, which has become more prevalent over the years. The environments we know today are littered with devices capturing and storing more and more information in digital format, and sometimes that becomes excessive and expensive to maintain.

BrightBalance+ is an iteration of a previous project – BrightBalance – that never came to real fruition. It consists of a smart system that focuses on light management in an attempt to automate light emission and intensity in a given household. The idea is to take the presence or absence of natural sunlight to, respectively, turn the lights off or on, raise or lower window blinds and so on.

Although the main purpose of the system is to manage lighting, BrightBalance+ seeks to monitor and react to temperature as well.

System Architecture

The architecture constitutes a **N** clients to **1** service relationship and splits the system into three different components, which interact with each other via Wi-Fi (IEEE 802.11 protocol):

- **Mobile application**
  Allows communication between users and the service, programmed using **Flutter** with **Android Studio** as the IDE;
- **Ubiquitous application**
  The combination of a microcontroller with sensors, actuators and a program, specialized to sensing the environment for phenomena that trigger context-aware actions. We use the **Arduino IDE** to control all hardware components;
- **Webserver**
  Database server to store and provide convenient access to system and user data. The webserver of choice, **Firebase**, will serve as the central point of BrightBalance+, storing user and sensor data, through **Firestore** database, and communicating with both the mobile and ubiquitous applications when needed.

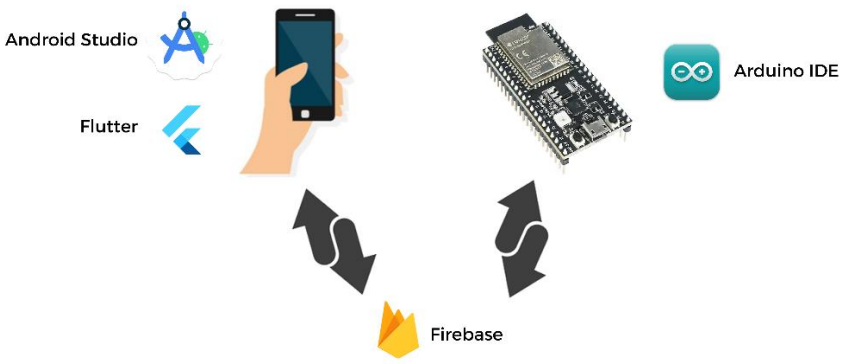

Figure 1 – System Architecture

Mobile Application

The BrightBalance+ mobile application will be implemented with Android Studio, using the Flutter framework.

In terms of functionality, users will be allowed to create their own profile to interact with the service. System specific user preferences, or “Moods” as designated in the application, to be stored in the webserver, will define properties like custom light colors and preferred light emission intensity to allow users to apply their own twist to the service. Moods will facilitate management across different rooms, as it will also be possible for users to create a representation of the divisions of their house in the application. Speaking of rooms, each will include an on-demand system on or off toggle, and display information about temperature and the amount of people currently in it.

Additionally, notifications will be sent to the user whenever the ubiquitous component is ready to take action regarding the raising and lowering of blinds and turning lights on and off, ultimately providing control to the user over BrightBalance+.

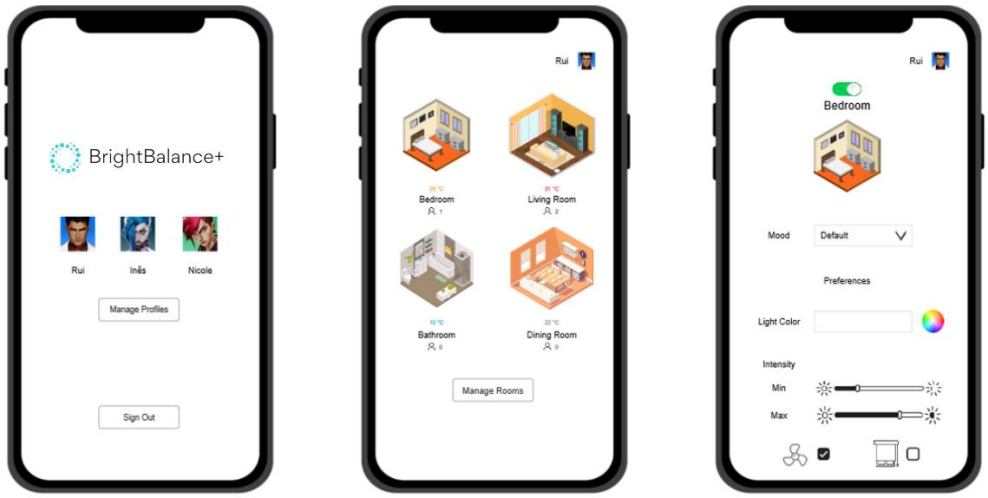

Figure 2 – BrightBalance+ Mockup

Ubiquitous Application

BrightBalance+ will require a microcontroller to interact with the environment of the user’s household. The module of choice will be the same ESP32 (see Fig. 3) used initially for BrightBalance.

Figure 3 – WiFi module, Espressif ESP32 S2 WROOM

The following Table 1 includes the sensors required to allow the correct behavior of the system, particularly in collecting information about its surrounding environments.

Sensor	Role	Image
LDR (Light Dependent Resistor) Photoresistors	Sensing presence or absence of light illuminating a room	
2x NTC (Negative Temperature Coefficient) Thermistors	Comparing interior with exterior temperature	

Table 1 – BrightBalance+ Sensors

The data gathered from photoresistors and thermistors will complement one another to enhance the system's behavior. The decision to use two thermistors instead of one was based on personal experiences. In Portugal, there are many buildings with no insulation which, when combined with other factors, makes divisions reach bothersome temperatures. During the summer, when sunlight beams through windows, the incoming heat gets trapped in the building interior, causing a greenhouse effect that increases indoor temperature. In contrast, during winter times, poorly insulated buildings lose heat which decreases indoor temperature. While BrightBalance+ does not offer a solution for indoor heat loss, the idea behind having two thermistors is to be able to sense indoor and outdoor temperatures simultaneously. In case a room starts reaching temperatures hotter than that of the outside, the system can suggest lowering the blinds and turning on the fan, two of BrightBalance+'s actuators found in Table 2.

Actuator	Role	Image
RGB LED (Light Emitter Diode)	Light emission based on environment conditions and active Mood	
Blinds	Room ambience based on sensor readings	
Fan (or other air conditioner system)	Room conditioning in case a room is too hot	

Table 2 – BrightBalance+ Actuators

Instead of using an actual fan and blinds, the ubiquitous system will use a green and a red LED to simulate the fan being turned on and off, respectively, and a servo motor to simulate the act of raising and lowering of the blinds.

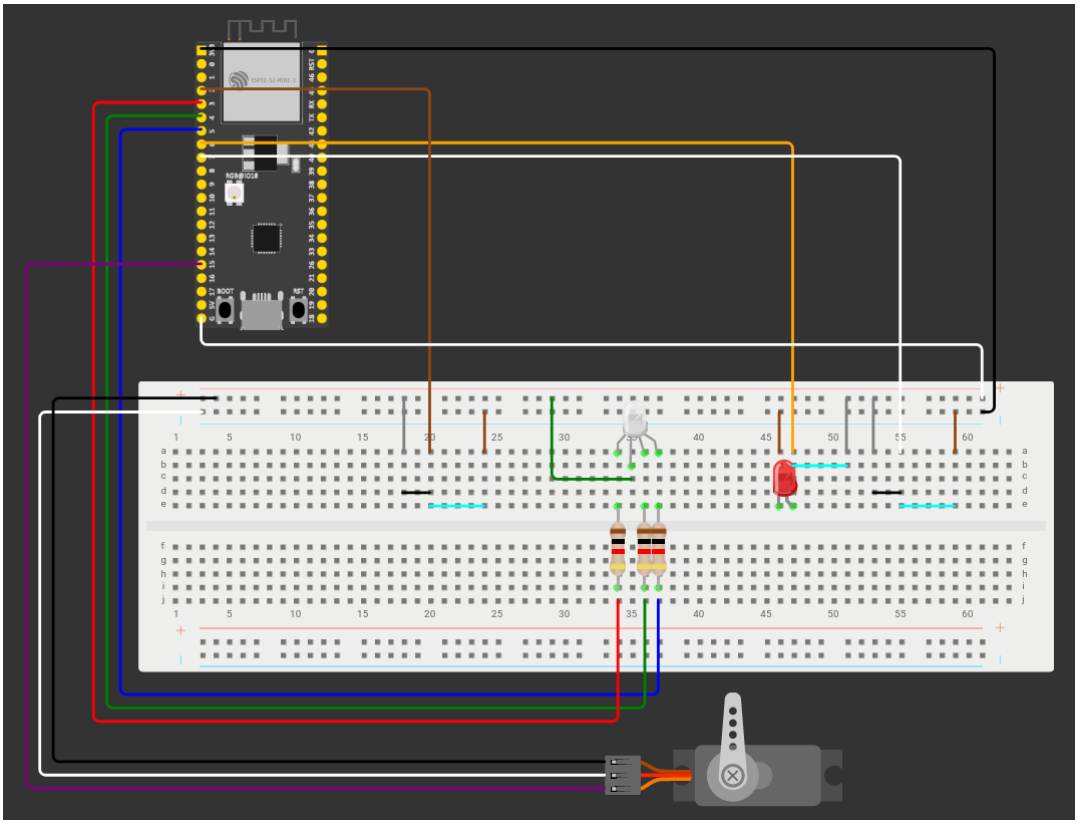

Figure 4 – Ubiquitous Architecture simulation

The simulation shown in Figure 4 and the system in Figure 5 (further ahead) do not include the LEDs that represent the fan due to, truthfully, forgetfulness. The BrightBalance+ team apologizes for this oversight, and will be carrying out the necessary implementation soon.

On a brighter note (haha 😀), because the editor used for the sketch does not include the same hardware materials as the ones used in the system, some of the elements in Figure 4 are only representative. Since the breadboard could not be flipped, the ground cable is connected to positive instead of negative, for example. With that in mind, the following are the so called representative elements:

- Ground cable connects to negative;
- Voltage cable connects to positive;
- 5-dot horizontal cable represents a 10k Ohms resistor;
- 3-dot horizontal cable corresponds to a thermistor;
- Represents a photoresistor.

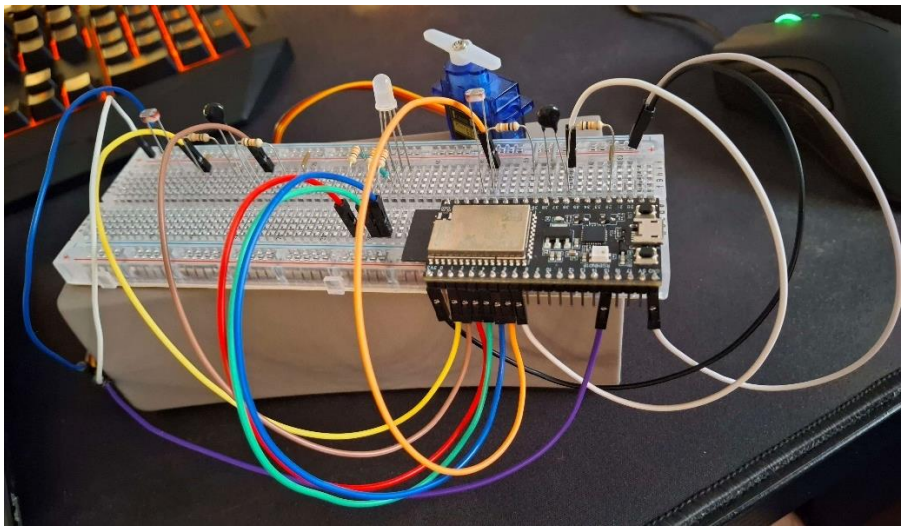

Figure 5 – Ubiquitous System

Note: Figure 5 is slightly misleading because it includes a second photoresistor (the one on the left side). This photoresistor was removed from the current version of the system due to implementation difficulties and subsequent discussion between BrightBalance+'s developers. From this discussion, it was concluded that a second photoresistor only played a role in enabling day and night modes. The photoresistor located indoors is already sufficient to sense the light levels in a room, whether that is through natural sunlight or the light emitted by the RGB LED. Additionally, because there is already the Moods feature that can provide more or less the same functionality, the team feels as though the second photoresistor is unnecessary.

Webserver

The webserver component plays a crucial role in the BrightBalance+ system, serving as the database server responsible for storing and providing convenient access to system and user data.

Firebase is selected as the webserver of choice for BrightBalance+ due to its suite of features tailored for real-time database management and seamless integration with various application platforms. Specifically, Firestore, a flexible and scalable database offered by Firebase, is utilized to store user and sensor data, ensuring data synchronization across client applications through real-time listeners.

As depicted in Figure 1 of this document, the Firebase server will serve as the central point of BrightBalance, storing user data (profiles, housing divisions, Moods, priorities, etc.) and monitoring data from the microcontroller (indoor and outdoor temperature, and number of people in each housing division). The server will be responsible for providing the necessary information for the actions performed by the mobile application and the operations executed by the microcontroller. This implies that, except for activating or deactivating the on-demand service, any communication between the mobile application and the microcontroller, or vice versa, will be routed through the server, which will receive and redirect the information to the intended component.

Current Functionalities

Until this point, the ubiquitous application was the point of focus as the means to understand what components were necessary and whether it was possible to make the connections initially planned. The current functions for the ubiquitous application are:

Function	Description
SetColor()	Modify the color of the emitted light
CalculateLightingAdjustment()	Capture photoresistor readings to calculate and adjust appropriate light emission brightness
GetIndoorsTemperature()	Get the indoor temperature
GetOutdoorsTemperature()	Get the outdoor temperature
AdjustBlinds()	Adjust the position of blinds

Table 3 – Ubiquitous Application functions

Future Work

Further development is still necessary for the ubiquitous application, as well as taking advantage of its Wi-Fi capabilities and the webserver. In terms of the mobile application, the team has already developed a few screens and established a connection between it and the webserver, but there are no defined functions. The idea is to provide the following functionalities:

Function	Description
SetProfile()	Create or modify a user profile
DisableProfile()	Remove a user profile
SignIn()	Verify user identity using a profile in the application
SignOut()	End the user's session in the application
CreateDivision()	Establish a new housing division
DeleteDivision()	Remove an existing housing division
Activate()	Enable a profile in a housing division, disabling in others
Deactivate()	Disable a profile in a housing division
CreateMood()	Establish a new Mood
EditMood()	Modify an existing Mood
DeleteMood()	Remove a Mood
GetTemperature()	Retrieve temperature data
GetUsers()	Obtain user data in housing divisions
EmitError()	Initiate an error notification for the system

Table 4 – Mobile Application functions